TITLE PAGE

Project Title: BudgetPal – A Bill Splitting and Expense Management Web Application

Team Members: Ayush Kumar Himamshu D
Devika C V
Sivaneshwari P
Vinodha G
Sasmitha K
Vaishaali B
Janaranjani C
Sam Daniel A
Aldrin Titus S
Mukilesh S

Guide/Mentor: Ms. Kuhoo Saxena

Institute/Organization: KIT-KalaignarKarunanidhi institute of technology,Coimbatore.

Date: 28/10/25

2.ABSTRACT

BudgetPal is a full-stack web application developed to simplify shared expense management among groups such as friends, roommates, and colleagues. The project solves the common problem of confusion and mismanagement in splitting bills and tracking group finances by providing an automated, transparent, and easy-to-use platform.

The application enables users to create groups, add expenses, split amounts automatically, and view individual balances in real time. It also provides a clear overview of transactions and settlements through an interactive dashboard.

BudgetPal is built using React.js for the frontend and Node.js with Express.js for the backend. The system uses PostgreSQL as its primary database, integrated with Supabase for efficient authentication, real-time data synchronization, and API management. JWT (JSON Web Tokens) ensures secure and reliable user session handling. The project is deployed on Vercel, offering fast and stable web hosting.

Technologies Used: React.js, Node.js, Express.js, PostgreSQL, Supabase, Vercel

Expected Outcome: A secure, modern, and scalable web application that automates group expense management, enhances transparency among users, and ensures effortless tracking of shared financial activities.

3.TABLE OF CONTENTS

4.INTRODUCTION

Background of the Problem

Managing shared expenses manually often leads to confusion, calculation errors, and miscommunication within groups. Traditional methods like notes, spreadsheets, or chats lack automation, resulting in disputes and inefficiency.

Objectives of the Project

- Automate expense tracking and bill splitting among group members.
- Provide a transparent and easy-to-use dashboard for financial summaries.
- Ensure secure data handling and user authentication.
- Support group creation, expense addition, and settlement tracking.

Scope of the Project

The system is built for small to medium-sized groups such as college friends, roommates, or office teams. It can later be expanded to

Significance / Use Cases

• Splitting expenses during trips or events.
• Managing monthly expenses among roommates.
• Tracking shared costs for college or office projects.
• Managing group contributions for parties or celebrations.
• Handling shared expenses in family budgets.
• Managing club or team event finances efficiently.

5. LITERATURE SURVEY / EXISTING SYSTEM

Existing solutions for managing shared expenses among groups, such as friends, colleagues, or roommates, have traditionally relied on manual methods or limited digital tools. The most common approach involves recording expenses in notebooks, using calculators, or updating shared spreadsheets. While these methods are simple, they lack automation, real-time updates, and an integrated way to track group balances. In many cases, communication through messaging applications or social media groups (like WhatsApp or Telegram) is used to keep track of who owes whom. However, these platforms are not designed for structured expense management and often lead to confusion, missed transactions, and miscommunication.

Some mobile applications, such as Splitwise and Tricount, have emerged as popular tools for expense splitting.Moreover, many of these platforms require premium features locked behind paywalls, which limits accessibility for all users. As a result, users seeking flexibility, transparency, and full control over their data are left without an ideal solution.

Furthermore, most available apps primarily cater to individual use cases and do not provide comprehensive visual analytics or detailed insights into spending behavior. This limits users' ability to understand their financial patterns and make informed decisions. The lack of centralized, customizable, and open solutions for collaborative financial tracking underscores the need for a more accessible and flexible platform like **BudgetPal**.

Limitations of Existing Solutions

The current landscape of group expense management tools presents several limitations that hinder efficiency, transparency, and user experience:

- **Inefficiency:** Manual expense tracking using spreadsheets or chat-based discussions is time-consuming and prone to arithmetic errors. The absence of real-time synchronization means members must manually update shared documents or constantly remind others about pending balances, resulting in delays and inconsistencies.

- **Lack of Transparency:** In traditional tracking methods, it's easy for expenses or contributions to go unrecorded, leading to mistrust among group members. The absence of an automated and clearly visualized system for viewing balances often leads to disagreements or uncertainty regarding individual dues.
- **Absence of Analytical Insights:** Most available systems focus solely on basic calculations and fail to provide meaningful analytics about spending behavior. Users cannot view categorized insights, monthly summaries, or expense trends, which are crucial for improving budgeting habits.

In summary, existing expense management methods whether manual or digital are limited by inefficiency, fragmentation, and lack of control. **BudgetPal** overcomes these limitations by providing a self-contained, customizable platform with real-time balance updates and secure backend integration.

6. PROPOSED SYSTEM

Project Architecture (High-Level Overview)

The proposed system, BudgetPal, is designed using the MERN-like architecture pattern but adapted to use PostgreSQL and Supabase for backend data handling. It follows a client–server model ensuring smooth interaction between frontend, backend, and database layers.

- **Frontend (React.js):**
  The frontend provides an intuitive, responsive, and user-friendly interface built with React.js. It enables users to perform all essential operations like creating groups, adding expenses, and viewing balances in real-time. The UI emphasizes clarity, allowing seamless navigation across modules.
- **Backend (Node.js + Express.js):**
  The backend serves as the application's control layer, processing requests from the frontend and interacting with the Supabase API. Built using Node.js and Express.js, it manages routing, handles logic for expense calculation, and ensures secure communication between the user and database.
- **Database (PostgreSQL via Supabase):**
  The system uses PostgreSQL as its primary relational database, hosted and managed through Supabase. Supabase acts as a backend-as-a-service, providing built-in authentication, database APIs, and real-time synchronization. This integration ensures scalable data handling, faster development, and enhanced security without complex configurations.

Features and Modules

Module	Key Features
User Authentication	Managed via Supabase's authentication service using email-based login and secure JWT tokens for session management.

Group Management	Users can create groups, invite members. Each group maintains independent expense records.
Expense Tracking	Allows adding shared expenses with automated split logic based on selected users or equal division.
Transaction Summary	Displays clear breakdowns of who owes whom, total balances, and group spending statistics.
Dashboard & Analytics	Provides users with an overview of recent activities, spending trends, and quick settlement insights.

Advantages Over Existing Systems

- **Simplified Expense Sharing:** Automates all calculations, reducing manual errors and confusion.
- **Cloud-Managed Database**: Supabase with PostgreSQL ensures reliability, real-time updates, and easy scalability.
- **Secure Authentication:** Replaces traditional password hashing with Supabase's built-in, token-based authentication system.
- **Responsive and Modern UI:** The React-based frontend delivers a seamless experience across devices.
- **Transparency and Collaboration:** Each group member can view shared transactions instantly, ensuring accountability and clarity.

7.SYSTEM REQUIREMENTS

Hardware Requirements

Component	Specification
Processor	Minimum Dual-Core Processor (Recommended: Quad-Core)
RAM	Minimum 4 GB (Recommended: 8 GB for smooth development and testing)
Storage	Minimum 10 GB free space for code, dependencies, and build files
Display	1366 × 768 resolution or higher for proper interface rendering
Network	Stable internet connection for API access and Supabase integration

Software Requirements

Category	Specification
Operating System	Windows 10/11, macOS, or Linux (Ubuntu preferred for deployment)
Frontend	**React.js** with JavaScript and modern UI libraries (e.g., Material UI / Tailwind CSS)
Backend	**Node.js (v18+)** with **Express.js** framework for RESTful APIs
Database	**PostgreSQL**, hosted and managed through **Supabase** for real-time operations and authentication
Version Control	**Git** with GitHub/GitLab for collaborative development and deployment
Package Manager	**npm** or **yarn** for managing dependencies
Development Environment	**Visual Studio Code** or equivalent IDE
Hosting (Optional)	Supabase (backend & DB), Render / Vercel (frontend) for cloud deployment
APIs / Tools	Supabase JS Client SDK for database and authentication handling
Browser Compatibility	Chrome, Edge, Firefox, Safari (latest versions)

8.SYSTEM DESIGN

The system is built on the **MERN-like architecture pattern**, but with **PostgreSQL and Supabase** replacing MongoDB and bcrypt. It follows the **MVC (Model-View-Controller)** pattern to maintain a clear separation between data handling, business logic, and user interface.

Architecture Flow:

1. **Frontend (View) — React.js**
    - Users interact through a responsive React-based interface.
    - Handles user actions such as logging in, posting lost/found items, searching, and viewing matches.
    - Communicates with the backend through RESTful API calls.

2. **Backend (Controller) — Node.js with Express.js**
    - Handles all core logic, routing, and communication between frontend and database.
    - Validates user inputs, manages session flow, and processes requests such as adding, updating, or fetching items.
    - Integrates with **Supabase** for authentication and database operations.
3. **Database (Model) — PostgreSQL via Supabase**
    - Stores structured data such as user details, item reports, notifications, and categories.
    - Managed through **Supabase**, providing built-in authentication, secure API access, and real-time capabilities.
    - Data relationships are efficiently handled using SQL schemas.

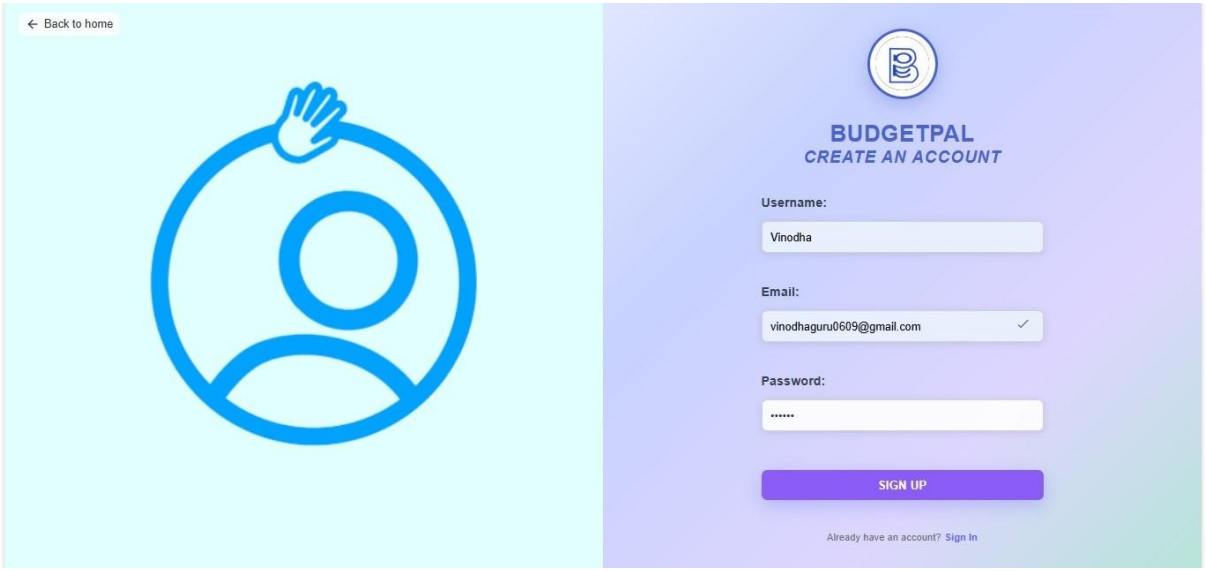
8.2 Data Flow

1. A user logs in or signs up through the frontend, which sends the credentials to the backend.
2. The backend interacts with **Supabase Auth** to verify and manage authentication.
3. Upon successful login, the backend grants access to protected routes and functions.
4. When a user adds a lost/found item, the data is sent via REST API to the backend.
5. The backend validates and forwards it to **Supabase (PostgreSQL)** for storage.
6. Users can search, filter, and view all items dynamically via API requests.
7. Notifications are triggered through backend logic when potential matches are found.

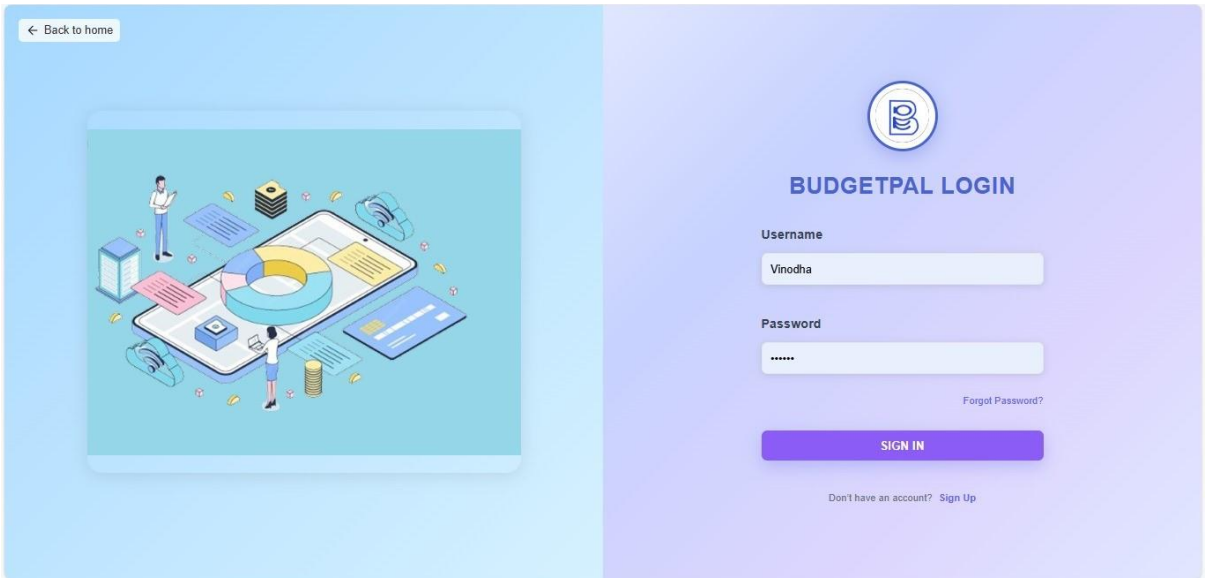
8.3 System Diagrams (Conceptual)

Use Case Diagram's

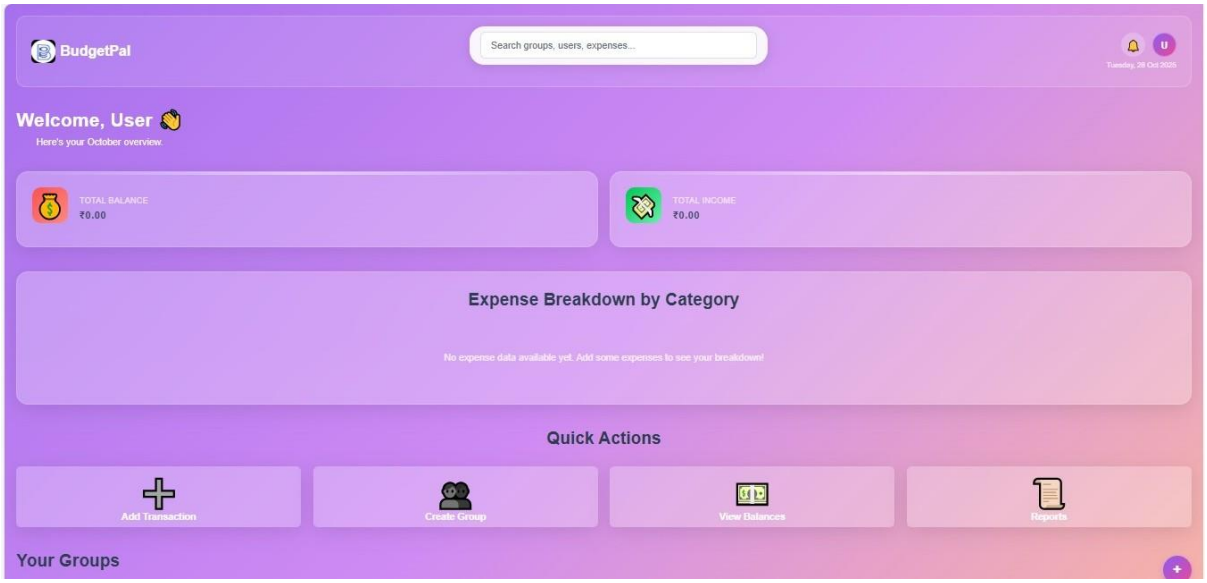
- **User SignUp Page**



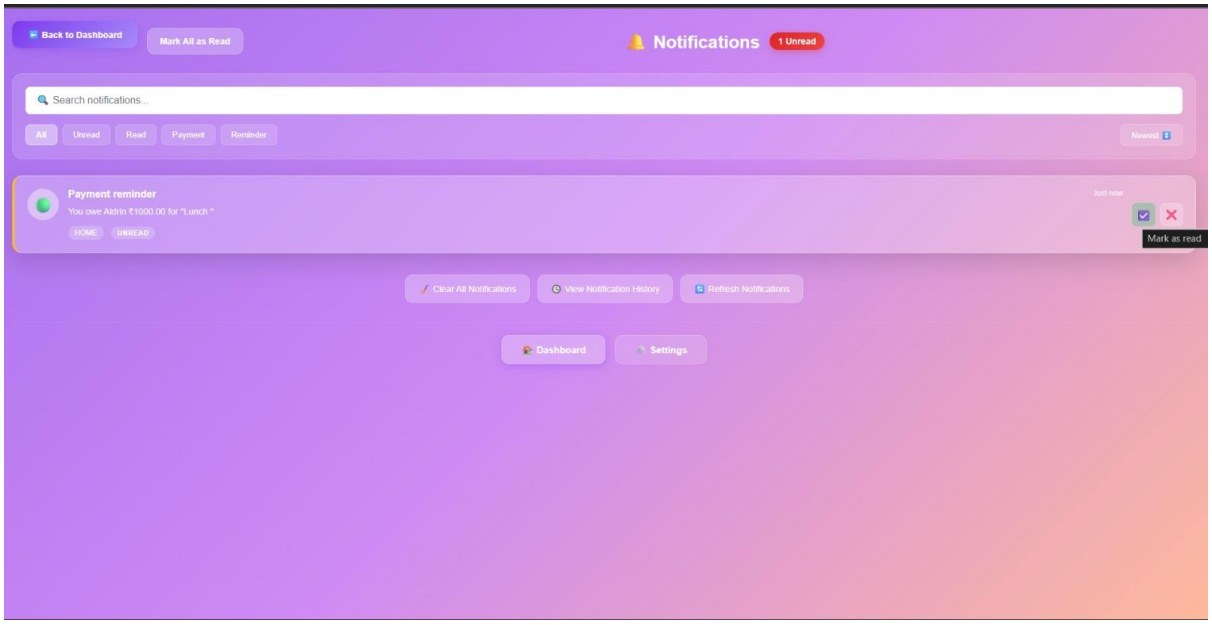
- **User Login Page**



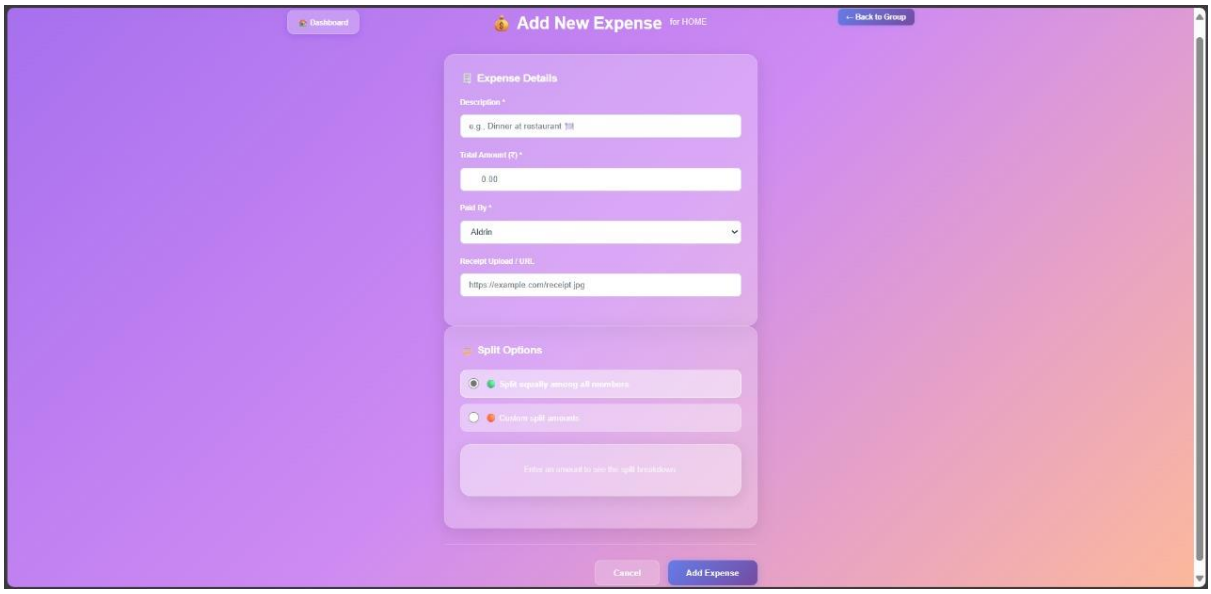
- **Dashboard**



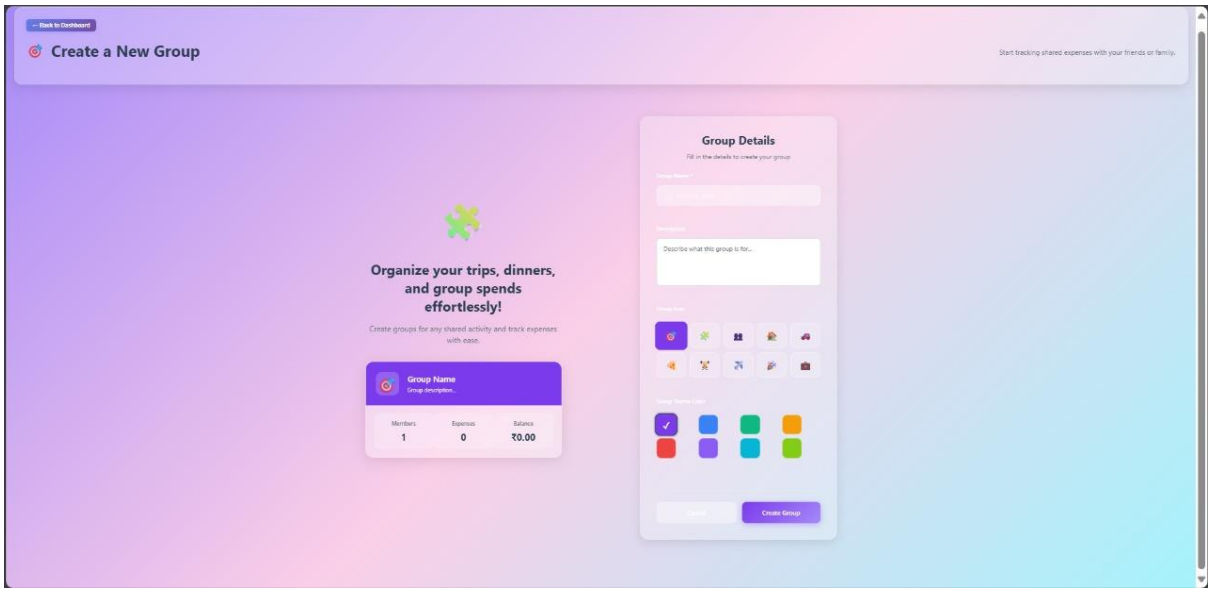
- **Notifications Page**



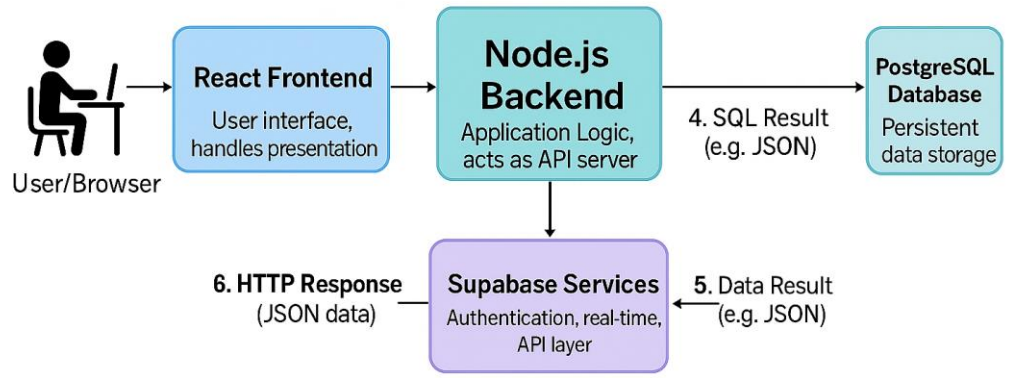
- **Add expense page**



- **Create group page**



Architecture Diagram (Simplified Flow)



8.4 Database Schema Overview

The PostgreSQL database consists of the following key tables:

Table Name	Description
users	Stores user details such as name, email, and authentication ID (managed by Supabase).
groups	Represents expense-sharing groups (like "Roommates", "Trip to Paris", etc.)
debts	Tracks who owes what to whom and Generated from expenses
expenses	Records individual expenses within groups.

8.5 Design Highlights

- **Real-time Data Updates** through Supabase's event listeners.
- **Secure Authentication** managed natively by Supabase (no manual hashing required).
- **Scalable Backend** using Express routes for clean API management.
- **Responsive UI** designed for both mobile and desktop views.

9. IMPLEMENTATION

The application's functionality relies on efficient backend logic rather than complex algorithms. The two main logical modules are Authentication Logic and Match Notification Logic.

A. Authentication Logic (via Supabase)

1. **User Registration:**
    - When a user signs up, the frontend collects the username,email and password.
    - Supabase Auth API creates a new user.
    - Upon successful verification, the user is added to the users table.
2. **User Login:**
    - Users log in using their registered credentials.
    - Supabase returns an access token and session details.
    - The frontend stores the session (token) securely and uses it for subsequent authenticated requests.

9.3 Example group routes (Backend)

```
import { Router } from 'express';
import { createGroup, addMemberToGroup, getGroupDetails, calculateGroupBalance,
getUserGroups, markDebtAsPaid, deleteGroup, getGroupMessages, sendGroupMessage }
from '../controllers/groupController.js';
import { authenticateToken } from '../middleware/auth.js';
import expenseRoutes from './expenseRoutes.js';

const router = Router();

// GET / - get all groups for the authenticated user
router.get('/', authenticateToken, getUserGroups);

// POST / - create a new group (requires authentication)
router.post('/', authenticateToken, createGroup);

// POST /:groupId/members - add a user to a group (requires authentication)
router.post('/:groupId/members', authenticateToken, addMemberToGroup);

// GET /:groupId - get details of a specific group (requires authentication)
router.get('/:groupId', authenticateToken, getGroupDetails);

// GET /:groupId/balance - get the simplified settlement plan (requires authentication)
```

```
router.get('/:groupId/balance', authenticateToken, calculateGroupBalance);

// POST /:groupId/mark-paid - mark a debt as paid (requires authentication)
router.post('/:groupId/mark-paid', authenticateToken, markDebtAsPaid);

// GET /:groupId/messages - get chat messages for a group (requires authentication)
router.get('/:groupId/messages', authenticateToken, getGroupMessages);

// POST /:groupId/messages - send a message to a group (requires authentication)
router.post('/:groupId/messages', authenticateToken, sendGroupMessage);

// DELETE /:groupId - delete a group (requires authentication)
router.delete('/:groupId', authenticateToken, deleteGroup);

// Mount expense routes under a group
router.use('/:groupId/expenses', expenseRoutes);

export default router;
```

9.4 Implementation Highlights

- Authentication handled entirely by **Supabase**, ensuring secure and scalable login.
- Real-time event handling for new item reports and notifications.
- API endpoints structured and modular using Express.js for maintainability.
- Reusable React components for consistency across all UI sections.

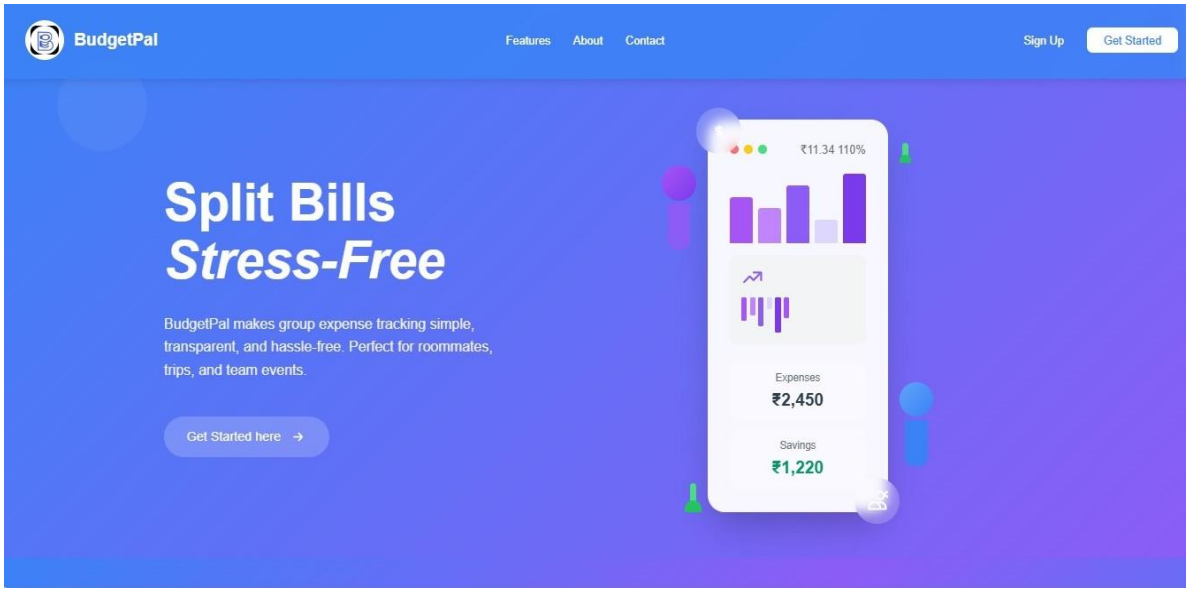
10. RESULTS AND DISCUSSION

10.1 Overview

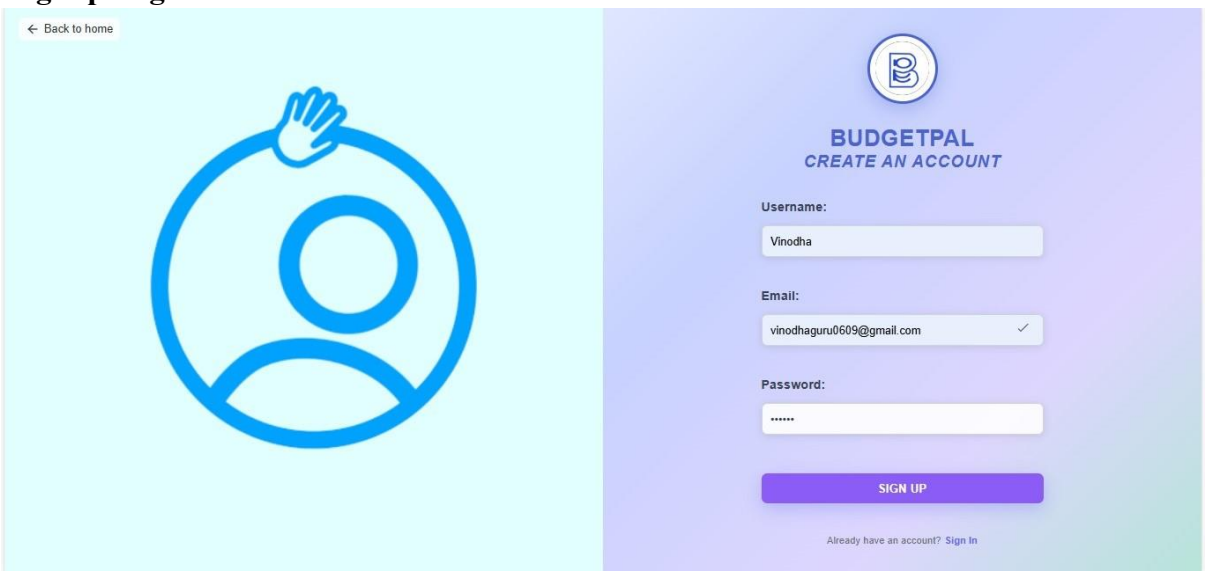
The final version of BudgetPal was successfully deployed on Vercel and verified through manual testing.
All key functionalities—user authentication, expense splitting, group creation, dashboard analytics, and settlements—worked smoothly with the PostgreSQL + Supabase backend and the React + Node.js + Express stack.

10.2 Screenshots of Application Outputs

1. **Home page**

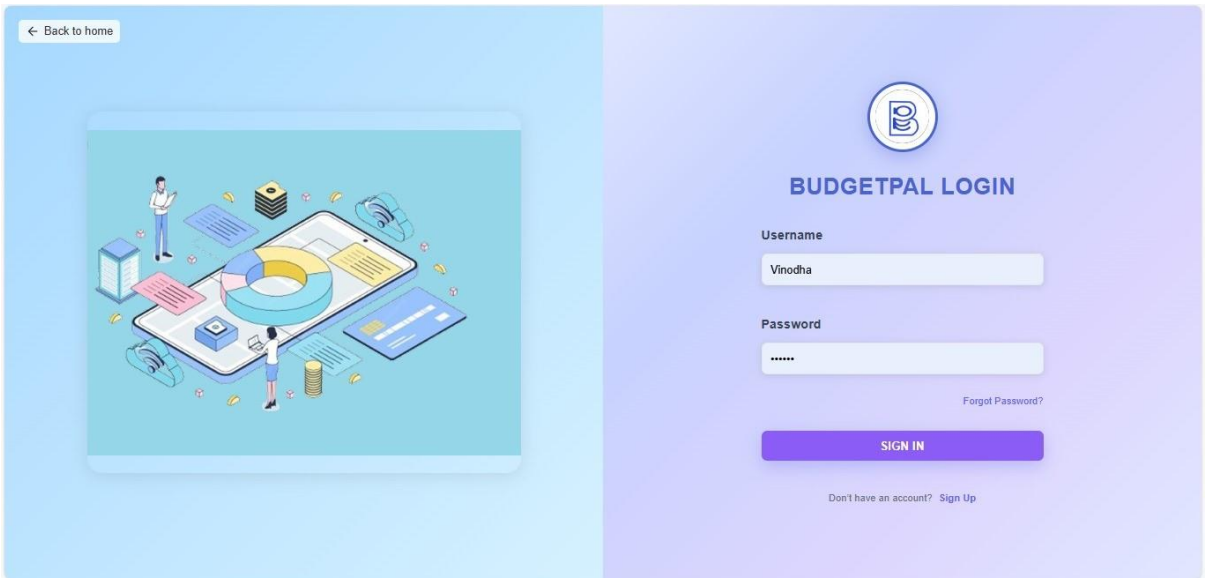


2. **Signup Page**

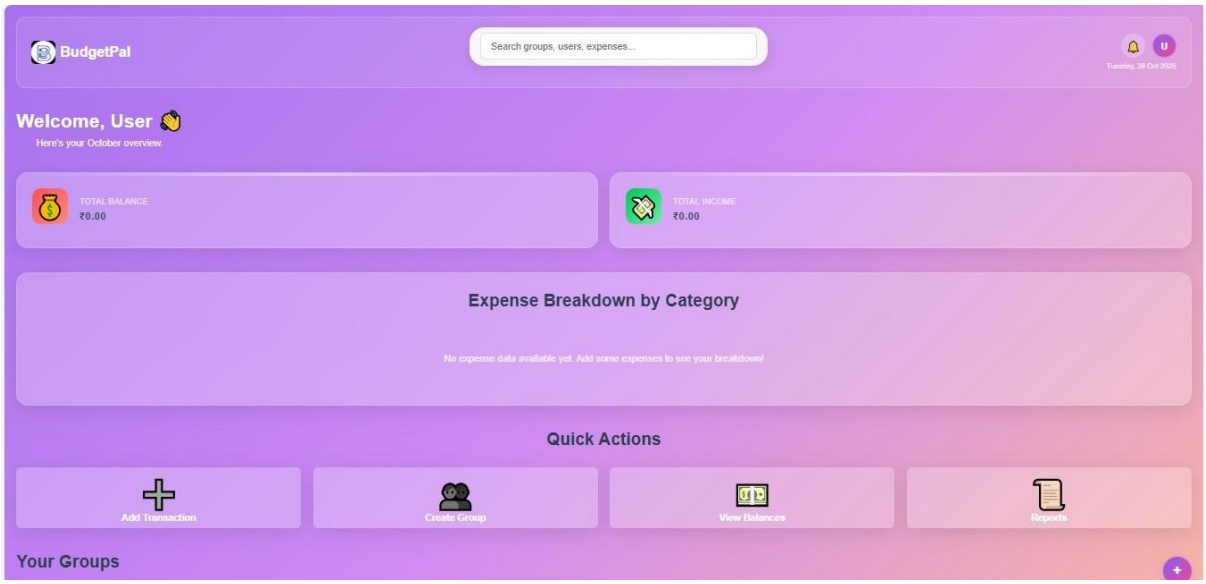


- Users can create new accounts or log in to existing ones.

3. Login Page

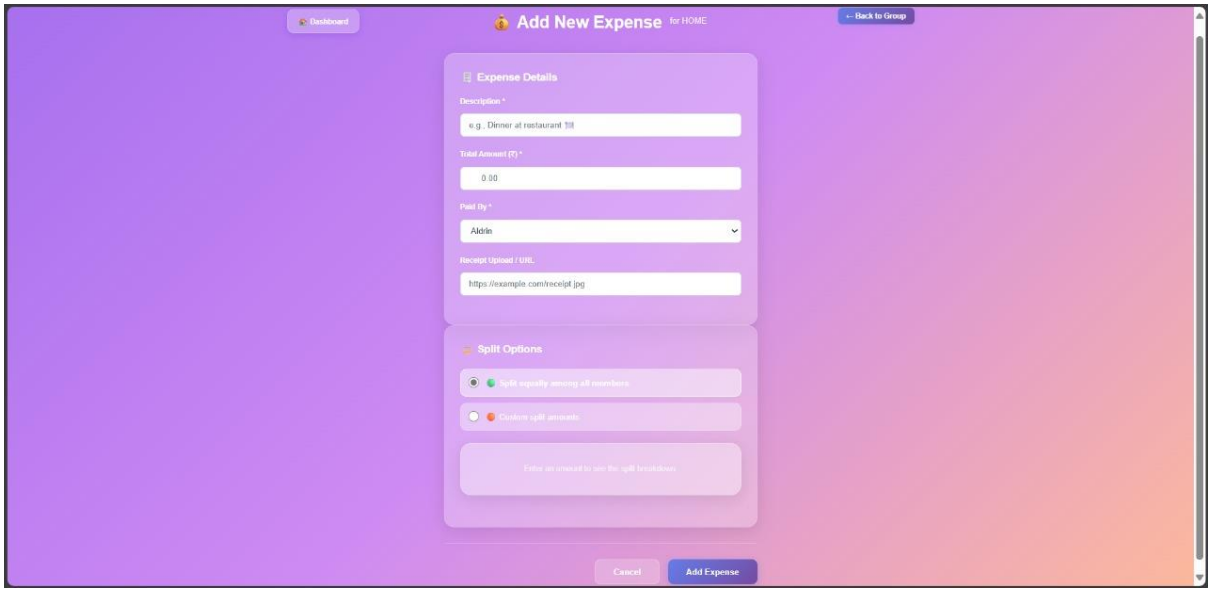


4. Dashboard



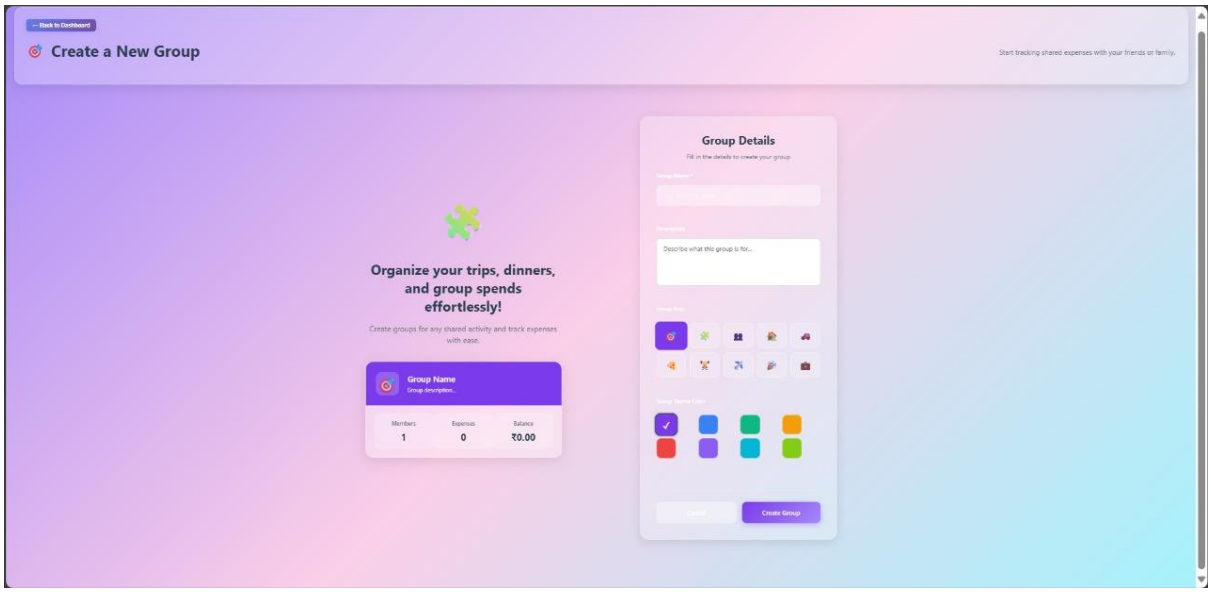
- Shows total balances, recent expenses, and settlements.
- Provides a clear summary of each user's owed and lent amounts.

5. Add Expense Page



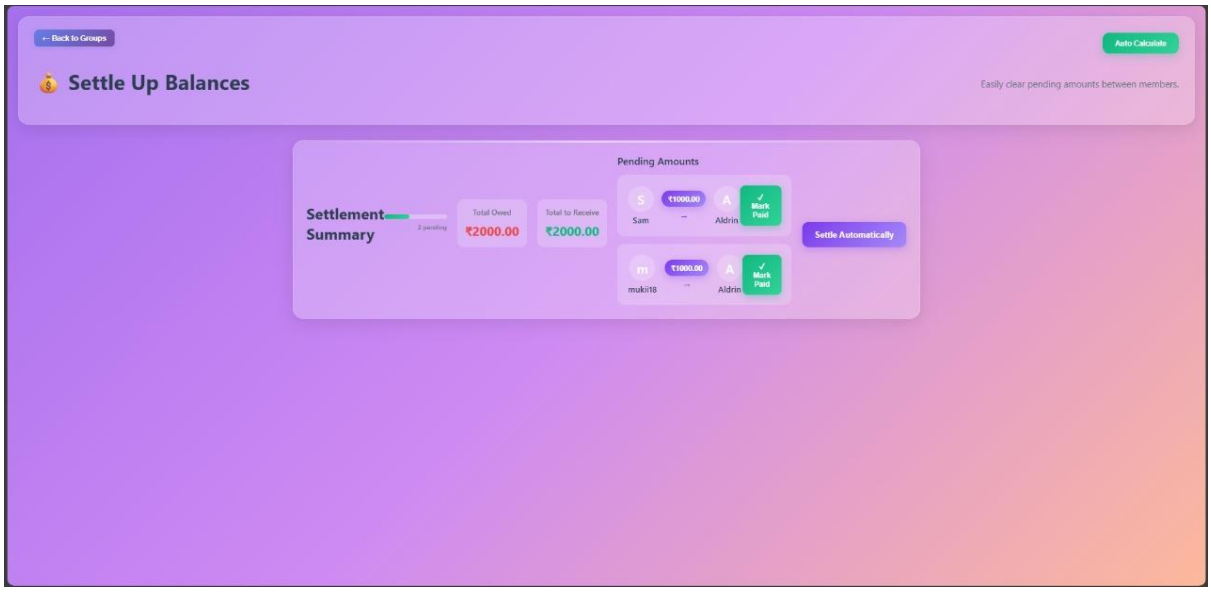
- Screenshot showing the "Add Expense" form where users enter the amount, select participants, and split costs.
- Confirms that data entered updates in real-time.

6. Group Creation Screen

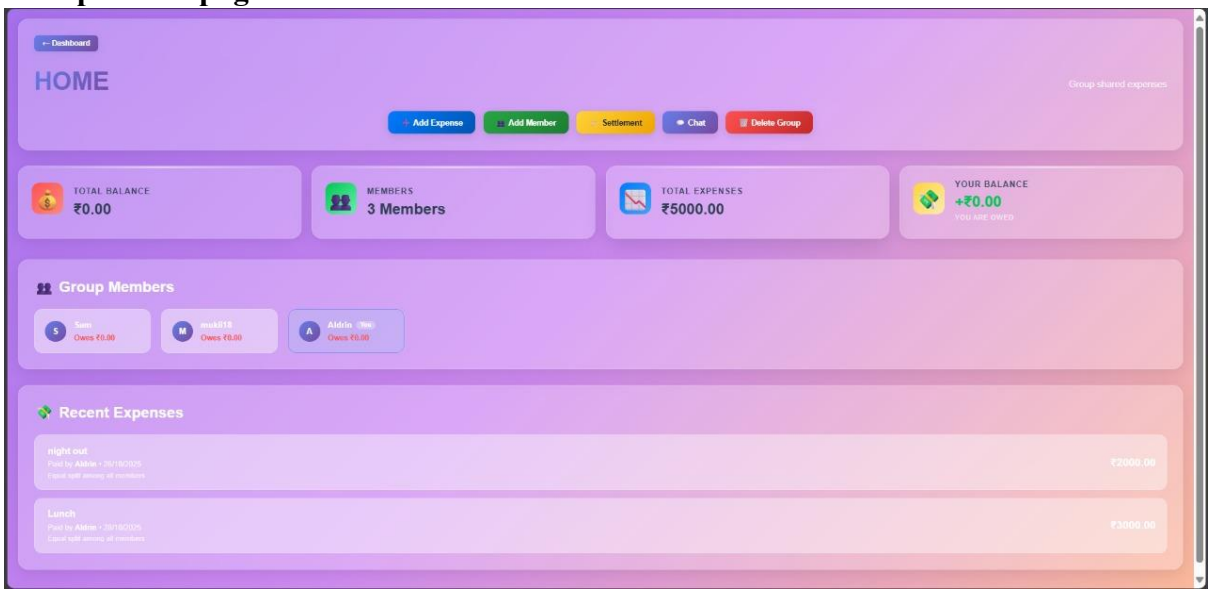


- Illustrates how users can create and manage multiple groups for different trips or activities.

7. View Settlements



8. Group details page



10.3 Comparison with Existing Solutions

Aspect	Traditional Method (Manual Splitting / Excel)	BudgetPal (Web App)
Expense Tracking	Manual entry and calculation errors	Automated calculations and sync
Group Management	Difficult to track who paid for what	Dedicated group module with member data
Settlement Updates	Time-consuming, manual	Instant real-time balance view
Accessibility	Local files or notes apps	Cloud-based, accessible anywhere
Data Security	No authentication	Secure Supabase auth and PostgreSQL storage

BudgetPal clearly provides a faster, more reliable, and collaborative way to manage shared expenses compared to manual methods.

10.4 Discussion

The overall testing and performance validation confirm that BudgetPal meets its objectives of providing a simple, efficient, and secure platform for managing shared expenses.
User interactions were smooth, database transactions were consistent, and the interface remained responsive even under multiple simultaneous inputs.
The real-time updates powered by Supabase further improved user experience by eliminating delays in data synchronization.

11. CONCLUSION AND FUTURE WORK

11.1 Achievements of the Project

The BudgetPal web application successfully delivers a complete and functional solution for efficient bill splitting and expense management among groups.
Developed using the MERN-style architecture (React, Node.js, Express) with PostgreSQL and Supabase integration, the system enables secure authentication, smooth data synchronization, and reliable backend operations.

Key achievements include:

- ✅ User Authentication: Implemented secure and reliable login/signup through Supabase Auth, ensuring data privacy and account integrity.
- ✅ Expense Splitting System: Enabled seamless creation of expenses, division among group members, and calculation of who owes or is owed.
- ✅ Group Management: Provided functionality for users to create, manage, and track multiple expense groups (e.g., trips, events, shared houses).
- ✅ Dashboard: Designed a responsive dashboard showing expense summaries, balances.
- ✅ Cloud Deployment: Successfully deployed the project on Vercel, ensuring accessibility across devices.

11.2 Limitations

While the project meets its core goals, certain limitations were identified during development and testing:

- ⚙️ Manual Testing Only: No automated testing framework has been implemented yet; all testing was performed manually.
- ⏱️ Real-Time Notifications: Users can view updated balances or settlements without even refreshing the page.
- 🌐 Single Deployment Environment: The project is hosted for demonstration on a single cloud platform (Vercel) and has not been optimized for high-scale concurrent users.

11.3 Possible Future Enhancements

Future improvements will focus on providing deeper financial insights to users through:

1. 🗓️ **Monthly Reports:**
   A feature to automatically generate a monthly summary of all expenses, balances, and settlements for each user and group.

2. 💰 **Spent Reports:**
   A detailed analysis showing how much each user spent, lent, and owes — helping users understand their spending habits and shared costs over time.

3. 💳 **Payment Gateway Integration**
   Enable secure and seamless payments directly within the platform, allowing users to settle balances instantly and manage transactions safely.

4. 💎 **Premium Features (Pricing Upgrade)**
   Introduce advanced analytics, customization options, and additional automation features under a premium plan to enhance the overall user experience and provide extra value.

5. 📧 **Automated Email Notifications**
   Implement an email notification system to send users reminders about pending payments, monthly reports, group settlements, and new feature updates — ensuring users stay informed and engaged.

6. 🤝 **Invitation / Referral System**
   Allow users to invite friends or group members via referral links or email invites, encouraging collaborative expense tracking and expanding the platform's user base.

Conclusion

In summary, BudgetPal achieves its objective of simplifying and automating group expense management in an intuitive and user-friendly manner.
The combination of React, Node.js, Express, PostgreSQL, and Supabase has resulted in a stable, scalable, and secure platform ready for real-world use and future expansion.

12. REFERENCES

1. React.js – *The Library for Web and Native User Interfaces*.
   Retrieved from: https://react.dev
2. Node.js – *JavaScript Runtime Built on Chrome's V8 Engine*.
   Retrieved from: https://nodejs.org
3. Express.js – *Fast, Uno*
4. *pinionated, and Minimalist Web Framework for Node.js*.
   Retrieved from: https://expressjs.com
5. PostgreSQL – *The World's Most Advanced Open Source Relational Database*.
   Retrieved from: https://www.postgresql.org
6. Supabase – *The Open Source Firebase Alternative for Building Secure Applications*.
   Retrieved from: https://supabase.com
7. Vercel – *Frontend Cloud Platform for Deployment and Scalability*.
   Retrieved from: https://vercel.com
8. MDN Web Docs – *Web Technologies Documentation for Developers (HTML, CSS, JavaScript)*.
   Retrieved from: https://developer.mozilla.org
9. GitHub Repository – *BudgetPal: Bill Splitting Web Application*.
   Retrieved from: https://github.com/ayush007-git/BudgetPal